Announcements

- Assignment 1
    - Will be posted on Wednesday, Jan. 9
    - Due Wednesday, Jan. 16
- Piazza
    - Please sign up if you haven't already
    - https://piazza.com/sfu.ca/spring2019/cmpt125
- Lecture notes
    - Posted just before class on the course website
    - https://coursys.sfu.ca/2019sp-cmpt-125-d1/pages/
- Final Exam
    - Apr. 10 at 12:00-15:00
    - Location TBA

C and Programming Basics

Lecture 2

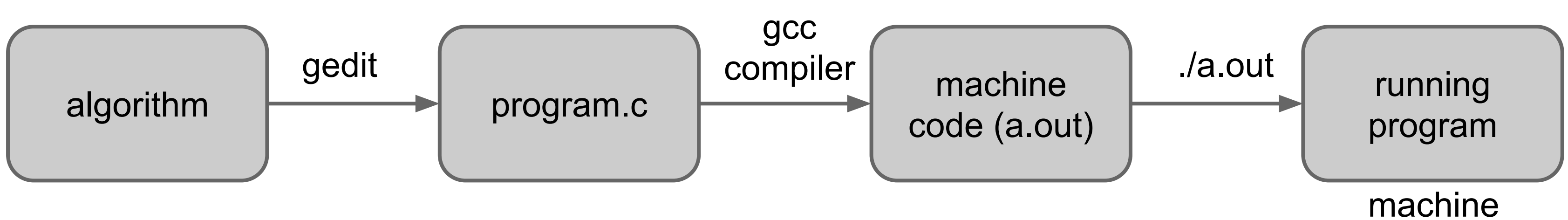
Today:

- The compilation process
- Differences between Python and C
- Variable declaration and strong typing
- The memory model: data vs. address

The Compilation Process

In Python:

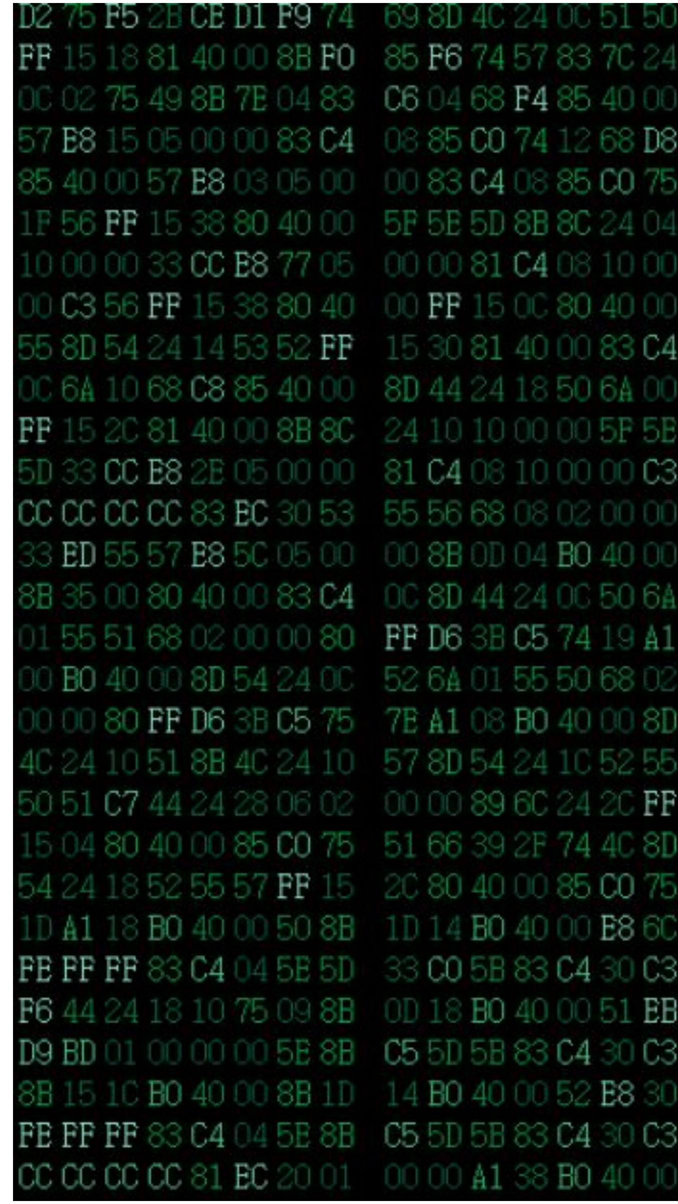
But in C, there is an extra step:



- Interpreter simulates one instruction at a time
- Compiler translates entire program in advance

Machine Language

- A *machine language* can be processed directly by a computer
  - A program is a sequence of instructions
  - Each instruction code (*opcode*) is represented by a number
  - No variable names or subroutine names in machine language!
- Each number is represented in binary
- Machine languages are very hard for humans to write and understand



Part of iTunes (trust me)

Machine Language

- Instructions: Operation codes, operands
    - Expressed as binary numbers

Memory	Operation code	Operand
??	1001 0010 Write to memory	0000 0010 2

Machine Language

- Instructions: Operation codes, operands
  - Expressed as binary numbers

Memory	Operation code	Operand
2	1001 0010 Write to memory	0000 0010 2

Machine Language

- Instructions: Operation codes, operands
  - Expressed as binary numbers

Memory	Operation code	Operand
2	1001 0010 Write to memory	0000 0010 2
5	1001 0111 Add to memory	0000 0011 3

Machine Language

- Instructions: Operation codes, operands
  - Expressed as binary numbers

Memory	Operation code	Operand
2	1001 0010 Write to memory	0000 0010 2
5	1001 0111 Add to memory	0000 0011 3

I made these up

Machine Language

- Binary numbers

BIN	DEC
0000	00
0001	01
0010	02
0011	03
0100	04
0101	05
0110	06
0111	07
1000	08
1001	09
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Machine Language

- Binary numbers

Carry			0	
	0	0	1	1
+	0	0	1	0
				1

BIN	DEC
0000	00
0001	01
0010	02
0011	03
0100	04
0101	05
0110	06
0111	07
1000	08
1001	09
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Machine Language

- Binary numbers

Carry		1	0	
	0	0	1	1
+	0	0	1	0
			0	1

BIN	DEC
0000	00
0001	01
0010	02
0011	03
0100	04
0101	05
0110	06
0111	07
1000	08
1001	09
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Machine Language

- Binary numbers

Carry	0	1	0	
	0	0	1	1
+	0	0	1	0
	0	1	0	1

BIN	DEC
0000	00
0001	01
0010	02
0011	03
0100	04
0101	05
0110	06
0111	07
1000	08
1001	09
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Machine Language

- Binary numbers

Carry	0	1	0	
	0	0	1	1
+	0	0	1	0
	0	1	0	1

XOR (add without carry)

IN	0	0	1	1
	0	1	0	1
OUT	0	1	1	0

BIN	DEC
0000	00
0001	01
0010	02
0011	03
0100	04
0101	05
0110	06
0111	07
1000	08
1001	09
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Machine Language

- Binary numbers

Carry	0	1	0	
	0	0	1	1
+	0	0	1	0
	0	1	0	1

XOR (add without carry)

IN	0	0	1	1
	0	1	0	1
OUT	0	1	1	0

AND (carry)

IN	0	0	1	1
	0	1	0	1
OUT	0	0	0	1

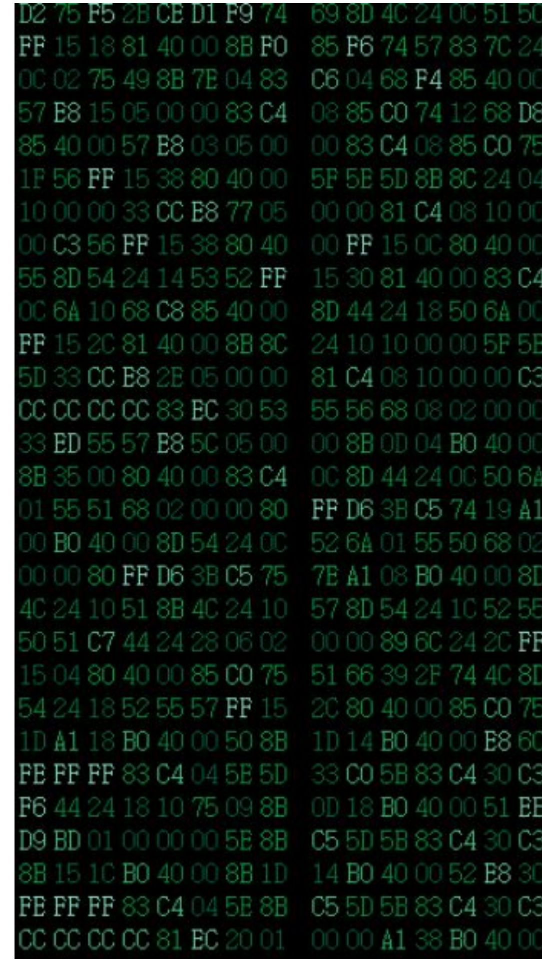
BIN	DEC
0000	00
0001	01
0010	02
0011	03
0100	04
0101	05
0110	06
0111	07
1000	08
1001	09
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Hexadecimals

- Binary: each digit has two possible values: 0 or 1
- Decimals: ten possible values per digit: 0-9
- Hexadecimal: each digit has sixteen possible values: 0-9, A-F
    - Convenient: one hexadecimal digit represents 4 binary digits (bits)

BIN	DEC	HEX
0000	00	0
0001	01	1
0010	02	2
0011	03	3
0100	04	4
0101	05	5
0110	06	6
0111	07	7
1000	08	8
1001	09	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Hexadecimals



- Binary: each digit has two possible values: 0 or 1
- Decimals: ten possible values per digit: 0-9
- Hexadecimal: each digit has sixteen possible values: 0-9, A-F
    - Convenient: one hexadecimal digit represents 4 binary digits (bits)

Assembly Language

- A high-level language compared to machine languages, but lower-level compared to C
- Abstract operation codes as mnemonics
- Abstract memory addresses as labels (variable names)
- An *assembler* translates mnemonics and labels into machine language

```
	.section	__TEXT,__text,regular,pure_instructions
	.globl	_main
	.align	4, 0x90
_main:                                  ## @main
	.cfi_startproc
## BB#0:
	pushq	%rbp
Ltmp2:
	.cfi_def_cfa_offset 16
Ltmp3:
	.cfi_offset %rbp, -16
	movq	%rsp, %rbp
Ltmp4:
	.cfi_def_cfa_register %rbp
	subq	$16, %rsp
	leaq	L_.str(%rip), %rdi
	movl	$0, -4(%rbp)
	movb	$0, %al
	callq	_printf
	movl	$0, %ecx
	movl	%eax, -8(%rbp)          ## 4-byte Spill
	movl	%ecx, %eax
	addq	$16, %rsp
	popq	%rbp
	ret
	.cfi_endproc

	.section	__TEXT,__cstring,cstring_literals
L_.str:                                 ## @.str
	.asciz	"Hello World!\n"


.subsections_via_symbols
```

Assembly for “Hello World!” program

Language Translators

- Python's *interpreter* speaks to the machine
  - translate and run instructions one at a time
- C code is *compiled* using gcc
  - translate all instructions before running
  - faster performance at run time
  - no interactive interpreter in C
- Programming languages are formal and lack the richness of human languages
  - If a program is nearly, but not quite, syntactically correct, then it will not compile
  - The compiler will not "figure it out"

Differences Between Python and C

- Python
  - print arg1, . . .
  - arg1 = raw_input()
  - int, float, str, bool
  - variables declared during execution
  - and, or, not
  - if-elif-else
  - for i in range(n)
  - indented blocks
  - lists may grow/shrink
- C
  - printf(format, arg1, . . .)
  - scanf(format, &arg1, . . .)
  - int, float, char
  - variables declared at compile time (strong vars)
  - &&, ||, !
  - if { } else if { } else { }
  - for (i = 0; i < n; i++) { }
  - { blocks in curly braces }
  - arrays are fixed in size

Variable Declaration

In C programs, you must declare your variables before using them.

```
int main ( ) {
    int a = 5;
    int b = 17;
    printf("The sum of %d + %d is %d\n", a, b, a+b);
}
```

The code `int a = 5` declares that you will use an `int`eger variable named “`a`”, which has an initial value of `5`

Strong Typing

- In C, you can’t change the type of a variable
    - Once an `int`, always an `int`
    - It is possible to change types in Python
- gcc reserves space for your variables in memory
    - Usually 4 bytes for an `int` or a `float`
    - Usually 8 bytes for a `long long` or a `double`
    - Usually 1 byte for a `char`
- The type of variable is important because all data is represented by 0’s and 1’s

Printing Characters

```
#include <stdio.h>

int main() {
  char c = ‘0’;
  printf(“%c\n”, c);
}
```

Output:

```
0
```

Printing Characters

```
#include <stdio.h>

int main() {
  char c = ‘0’;
  printf(“%c\n”, c);
}
```

Interpret the variable c as a character

Output:

```
0
```

Printing Characters

```
#include <stdio.h>

int main() {
  char c = ‘0’;
  printf(“%d\n”, c);
}
```

Interpret the variable c as a **decimal integer**

Printing Characters

```
#include <stdio.h>

int main() {
  char c = ‘0’;
  printf(“%d\n”, c);
}
```

Interpret the variable c as a **decimal integer**

Output:

48

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Memory Model

- Each var stored in a unique memory location
  - its *address*
  - represented by a number (an integer, usually)
- Thus a variable is composed of 3 things:
  - its type
  - its value
  - its address
- Some programs need the address explicitly
  - Addresses can be stored in variables
    - A variable that contains the address of another variable is called a *pointer*

Using an Address

```
#include <stdio.h>

int main ( ) {
    int a = 0, b = 0;
    scanf("%d", &a);
    scanf("%d", &b);
    printf("The sum of %d + %d is %d\n", a, b, a+b);
}
```

- The input function `scanf()` needs to know *where* to store the input integer
- “`&a`” represents the address of `a`

Acknowledgement

These slides are the work of Brad Bart (with minor modifications)